Time dependent policy-based access control

Vasilikos, Panagiotis; Nielson, Flemming; Nielson, Hanne Riis

Published in:
Leibniz International Proceedings in Informatics

Link to article, DOI:
10.4230/LIPIcs.TIME.2017.21

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Time Dependent Policy-Based Access Control*

Panagiotis Vasilikos[1], Flemming Nielson[2], and Hanne Riis Nielson[3]

1 **Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark**
panva@dtu.dk
2 **Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark**
fnie@dtu.dk
3 **Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark**
hrni@dtu.dk

Abstract

Access control policies are essential to determine who is allowed to access data in a system without compromising the data's security. However, applications inside a distributed environment may require those policies to be dependent on the actual content of the data, the flow of information, while also on other attributes of the environment such as the time.

In this paper, we use systems of Timed Automata to model distributed systems and we present a logic in which one can express time-dependent policies for access control. We show how a fragment of our logic can be reduced to a logic that current model checkers for Timed Automata such as UPPAAL can handle and we present a translator that performs this reduction. We then use our translator and UPPAAL to enforce time-dependent policy-based access control on an example application from the aerospace industry.

1998 ACM Subject Classification D.4.6 Security and Protection, D.2.4 Software/Program Verification, C.2.4 Distributed Systems, F.4.1 Mathematical Logic

Keywords and phrases Access Control, Timed Automata, Time-Dependent Policies, UPPAAL

Digital Object Identifier 10.4230/LIPIcs.TIME.2017.21

1 Introduction

Motivation. Cyberphysical systems play an increasingly important role in the technology development in many industries such as the aerospace, the automotive and the medical. Embedded systems are key components to cyberphysical systems and while verifying their safety goals has received a significant focus until now, security has been left for later. As more and more cyberphysical systems are integrated with real-time hardware, complex software, and internet connected devices through wireless connections, ensuring security goals of those systems is becoming essential. Particularly, assuring the confidentiality or the integrity of the information manipulated by the different components of a cyberphysical system is a crucial security goal.

Information security is usually achieved by access control policies, which formally specify desired flows of information inside a system. Access requests to the resources/data (objects)

* The authors are supported in part by the IDEA4CPS Reseearch Centre studying the Foundations for Cyber-Physical Systems and granted by the Danish Research Foundation for Basic Research (DNRF86-10).

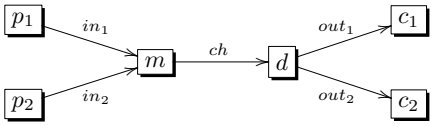


Figure 1 The processes and channels of the gateway example.

of the system by users (subjects) are then either denied or allowed by a monitor that enforces the access control policies. The literature offers a vast number of access control models, where among all, the most used in practice are the discretionary access control (DAC) [29], mandatory access control (MAC) [28] and role-based access control (RBAC) [12], while lately a great attention has been given to the attribute-based access control (ABAC) model [17], wherein access control may depend on the attributes of the accessed data or the attributes of the environment such as the time.

Although access control policies is a well-established approach for information security at the subject-object level, distributed systems require precise policies that express also the desired information flows that occur at the application level, such as *explicit* flows. As an example, consider the explicit flow from the variable y to the variable x that arises from the assignment $x := y$ and the access control policy "x can only be modified by p", where p is a trusted process in the system. Although the assignment $x := y$ executed by p does not violate the access control policy of the system, the fact that p resides in a distributed environment together with potential bugs inside its source code gives no guarantees that the value of y was written by p. For instance, the value of y could have been influenced by an untrusted process p' after a communication between p and p' and consequently p' would have also influenced the value of x. To see more challenges that arise inside distributed systems consider the work of [25] where it illustrates that security policies may need to depend on the actual content of the data, while ABAC [17] models also address the need for time-dependent security policies.

Contribution. It is natural then to extend the enforcement of safety properties of embedded systems with enforcement of access control policies. The idea is that having an abstract model of an embedded system, one could eliminate possible security violations in the trusted part of the system before the actual run of the system happens. We use *Timed Automata* [3, 1] to model distributed systems and we specify an information flow instrumented semantics that allows us to record information about the accesses being performed; we call this information a behaviour of the system. To deal with formal definitions of security policies we present a behaviour logic (based on the behaviours of the system) that supports the specification of content, time and information dependent access control policies. Verification of Timed Automata has been successfully achieved by model checkers based on the timed computation tree logic (TCTL) [2], and consequently, we propose a reduction of a substantial fragment of our logic to a logic that can be handled by the well-established model checker UPPAAL [30]. Finally, we present a translator that performs this reduction and we illustrate our development using an example from the aerospace industry. Figure 1 sketches the example: *A gateway with two processes, each of them produces data for different targets, uses a multiplexer and a demultiplexer to successfully deliver the data to the intended target. The multiplexer merges the data from the producers and sends it to the demultiplexer who is responsible for delivering it to the right target. The target of the data depends both on the time of the system while also on the content of it and thus it is challenging to express the appropriate security policy.* The example is based on the secure gateway presented in [22], where a seperation

kernel is used to allow the accesses to the resources of the system by the system's processes to be *temporal* (based on time) and based on an *information flow policy*. The seperation of the resources is used to ensure that untrusted processes such as passenger's devices can have access to onboard communication systems, without alerting the safe operation of the aircraft.

Related work. There are many other papers dealing with access control [27, 19, 15, 14, 34, 32], however without considering time-dependent security policies; a survey of access control models is available at [11].

A rich logic that allows reasoning about time-dependent policies, together with a proof checker for the logic is considered in [10], however there are no information flow considerations at the application level such as explicit flows. SecPAL [5] is another logic that supports time-dependent policies, as well as the encoding of many well-known policy idioms such as DAC, MAC, RBAC, and ABAC, however the enforcement of time constraints is external to the language. A somewhat different approach has been taken in [4], where a monitor is used to enforce time-dependent access control, by checking a system's logs that records the different actions of the users in a database system. Our contribution focuses on the challenges of time-dependent access control for embedded systems modeled as Timed Automata.

The work of [20] presents a formal specification and verification of the temporal role-based access control (TRBAC) model [6], a flexible model in which the roles of the users of the system are enabled or disabled depending on the time of the system. They then use UPPAAL [30] to model a TRBAC system and verify the desired security policies. The same authors of this paper, present in [21], an extension of this model which is based on the generalized-TRBAC (GTRBAC) model [18]. The work of [13] considers spatial-TRBAC (STRBAC) models [7] in which the rights of the user may depend on the time as well as on the location of the user; again the different roles of the system are modeled as timed automata and verified in UPPAAL. Carlo Combi et.al in [9] merges temporal role-based access control with workflows, while in [8] he defines access-controlled temporal networks, an extension of the conditional simple temporal networks with uncertainty which allows you to model users and temporal authorization constraints. Although all of those models deal with an important number of access control policies at the subject-object level considering time dependencies, they are not able to express time-dependent policies with information flow considerations that occur at the application level of the system (e.g does a process running on behalf of a user respects the access control policy?).

The work of [26] formalizes the timed decentralised label model (TDLM) an extension of the traditional and well-established decentralised label model (DLM) [23], which deals with both information flow and time-dependent security policies; however, their work does not consider an enforcement mechanism for the policies. Our key contribution is to develop a logic that allows the specification of time, data's content and information flow dependent policies for access control, and to make use of current model checkers such as UPPALL [30] for the enforcement of the policies.

Organisation. The remainder of this paper is organized as follows. In Section 2, we give the definition of a Timed System (a system of Timed Automata) and in Section 3 we define an information flow instrumented operational semantics for Timed Systems. Section 4 presents the syntax and the semantics of our behaviour logic called **BTCTL** (behaviour TCTL) and we illustrate how we can successfully express security policies for our gateway example. In Section 5 we present the reduction of the **BTCTL** logic to a variation of the TCTL [2], called $\mathbf{TCTL}^+$ and in Section 6 we present our translator. Finally, in Section 7, we give our

conclusions and we outline our future work, while in appendix A we give the proof of our main theorem.

2 Systems of Timed Automata

2.1 Timed Systems

A *Timed System TS*

$$p_1 : \mathsf{TA}_1 \,||\, ... \,||\, p_n : \mathsf{TA}_n \ (n \geq 1)$$

which sometimes we will call system, is the parallel composition of n timed automata called processes, written as $TS{=}(TA_i)_{i \leq n}$.

The processes are able to exchange information via synchronous message passing, using polyadic channels from the finite set **Chan**. Each of the processes in the timed system, is labelled with a unique identifier $p \in \mathbf{P}{=}\{p_1, ..., p_n\}$ and we write $\mathbf{Var}_p$ and $\mathbf{Clock}_p$ for the data variables and clocks appearing in the process p. We also require that the sets of data variables and clocks for the processes are mutually disjoint ($\forall i \neq j : \mathbf{Var}_{p_i} \cap \mathbf{Var}_{p_j} = \emptyset \wedge \mathbf{Clock}_{p_i} \cap \mathbf{Clock}_{p_j} = \emptyset$) and we write $\mathbf{Var}{=}\bigcup_i \mathbf{Var}_{p_i}$ and $\mathbf{Clock}{=}\bigcup_i \mathbf{Clock}_{p_i}$ for the overall data variables and clocks apperaring in the timed system.

2.2 Timed Automata

Formally, we model a *Timed Automaton* [3, 1] TA as a 4-tuple $(q_\circ, \mathsf{E}, \mathsf{I}, \mathsf{Q})$ where $q_\circ$ is the initial location of the automaton, E is a finite set of edges, I is mapping from the automaton's locations to conditions that impose invariants, and Q is the set of the automaton's locations.

The edges are labelled with actions $g \to act{:}\ \vec{r}$ and take the form $(q_s, g \to act{:}\ \vec{r}, q_t)$ where the syntax of the *act* is given by

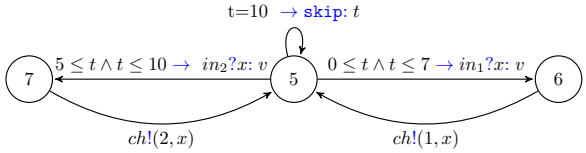
$$act ::= \vec{x} := \vec{e} \mid ch!\vec{e} \mid ch?\vec{x}$$

and $q_s \in \mathsf{Q}$ is the source location and $q_t \in \mathsf{Q}$ is the target location.

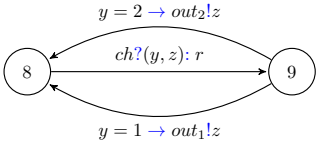
Every action $g \to act{:}\ \vec{r}$ consists of a guard g which has to be true in order for the action to be performed and it ends with a reset on the clocks $\vec{r}$. The assignment action $g \to \vec{x} := \vec{e}{:}\ \vec{r}$ performs multiple assignments $\vec{x} := \vec{e}$, while the action $g \to ch!\vec{e}{:}\ \vec{r}$ is used to communicate the data of the expressions in $\vec{e}$ using the channel ch and the action $g \to ch?\vec{x}{:}\ \vec{r}$ is used to receive data and store it in the variables of the vector $\vec{x}$. We shall assume that the sequences $\vec{x}$ and $\vec{e}$ of data variables and expressions, respectively, have the same length and that $\vec{x}$ does not contain any repetitions. Finally, we write $\vec{x}(i)$ (and also $\vec{e}(i)$) for the i-th element of the vector $\vec{x}$ (and $\vec{e}$ respectively). To cater for special cases of the assignment action, we shall allow to write $g \to \texttt{skip}{:}\ \vec{r}$ when $\vec{x}$ (and hence $\vec{e}$) is empty; also for any kind of action we shall allow to omit the guard g when it equals to tt and to omit the clock resets when $\vec{r}$ is empty. If it is the case that all of the above take place together we omit the whole action.

▶ **Example 1.** The timed system of our gateway example is given in Figure 2. The timed system consists of six processes $\mathbf{P} = \{p_1, p_2, m, d, c_1, c_2\}$. Two producers p_1 and p_2 send their data via the channels in_1 and in_2 respectively. The multiplexer m collects the data from the producers using the channel ch and then forwards it to the demultiplexer d, who then distributes it to the consumers c_1 and c_2 via the channels out_1 and out_2 respectively. The access policy that we want to impose here is that the consumers c_1 and c_2 read data only from the producers p_1 and p_2, respectively.

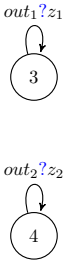
(a) The producers p_1 (top) and p_2 (bottom)



(b) The multiplexer m



(c) The demultiplexer d



(d) The two consumers c_1 (top) and c_2 (bottom)

Figure 2 The timed system for the gateway example.

We use clocks to model the temporal accesses to the resources of the system as required in [22]. In particular we use two clocks v (in the multiplexer) and r (in the demultiplexer) to model instantaneous transitions (time does not pass) and we use a clock t (in the multiplexer) to split the overall execution time of the timed system into periods of 10-time units. In each period the multiplexer m reads data from the channel in_1 only whenever $t \in [0, 7]$, while it reads data from the channel in_2 only whenever $t \in [5, 10]$. Whenever $t \in [5, 7]$ the multiplexer chooses non-deterministically to read either from in_1 or in_2. The multiplexer then transports the data together with a constant, using the dyadic channel ch to the demultiplexer d; the constant is used as a mark to indicate the source of the data. Finally, the demultiplexer delivers the data to the right consumer according to the constant.

The expressions e, guards g and conditions c that label the locations are defined as follows using boolean tests b:

$$
\begin{array}{lcl}
e & ::= & e_1 \,\mathsf{op}_a\, e_2 \mid x \mid n \\
b & ::= & \mathsf{tt} \mid \mathsf{ff} \mid e_1 \,\mathsf{op}_r\, e_2 \mid \neg b \mid b_1 \wedge b_2 \\
g & ::= & b \mid r \,\mathsf{op}_c\, n \mid (r_1 - r_2) \,\mathsf{op}_c\, n \mid g_1 \wedge g_2 \\
c & ::= & b \mid r \,\mathsf{op}_d\, n \mid (r_1 - r_2) \,\mathsf{op}_d\, n \mid c_1 \wedge c_2
\end{array}
$$

The arithmetic operators op_a and the relational operators op_r are as usual. For comparisons of clocks we use the operators $\mathsf{op}_c \in \{<, \le, =, \ge, >\}$ in guards and the less permissive set of operators $\mathsf{op}_d \in \{<, \le, =\}$ in conditions.

3 Information Flow Instrumented Semantics

3.1 Behaviours

The transitions of the timed systems are labeled with behaviours. A *behaviour* records information relevant to the action that has occurred and also information about the processes

that were involved in the action. Formally a behavior takes the form

$$\mathfrak{b} \in \mathcal{B}_{local} \cup \mathcal{B}_{com}$$

where

$$\mathcal{B}_{local} = \mathbf{P} \times \overrightarrow{\mathbf{Var}} \times \overrightarrow{\mathbf{Exp}}$$

are the behaviours that occur due to assignments and

$$\mathcal{B}_{com} = \mathbf{P} \times \mathbf{Chan} \times \overrightarrow{\mathbf{Var}} \times \overrightarrow{\mathbf{Exp}} \times \mathbf{P}$$

are the behaviours that occur due to the communication between two processes. We write $\overrightarrow{\mathbf{Var}}$ and $\overrightarrow{\mathbf{Exp}}$ for the sets of vectors with elements over the data variables and arithmetic expressions respectively.

For instance, the local behaviour $p : (\vec{x}, \vec{e})$ records that the process p has performed an assignment in which the vector $\vec{e}$ is used to modify the variables of the vector $\vec{x}$, while the behaviour $p : ch(\vec{x}, \vec{e}) : p'$ records that a communication between the processes p (the sender) and p' (the receiver) has happened, using the channel ch, and the vector $\vec{e}$ is the vector of expressions whose values have been communicated and have been bound to the variables of the vector $\vec{x}$.

In all of the behaviours, the vectors that are being used must have the same length while for the delay action we will write the empty behaviour ϵ.

3.2 Operational Semantics

To specify the semantics of timed systems, let σ be a state mapping data variables to values (which we take to be integers), let δ be a clock assignment mapping clocks to non-negative reals and let κ be a mapping from data variables to sets of processes which we will call *writers*. The mapping κ is used to monitor the explicit flows that occur from the assignments and the communication between two processes in the system; we explain the use of κ in more detail in a while. We then have total semantic functions $[\![.]\!]$ for evaluating the expressions, boolean tests, guards, and conditions; we evaluate expressions either with a state σ or the mapping κ, where for the first case the evaluation returns a value and in the second it returns the writers of the expression. The evaluation of boolean expressions only depends on the states, whereas that of guards and conditions also depend on the clock assignments.

The configurations of a timed system $TS = (TA_i)_{i \leq n}$ are of the form $\langle \vec{q}, \sigma, \delta, \kappa \rangle$, where $\vec{q}$ is a vector of nodes and we write $\vec{q}(i)$ for the i-th element of the vector $\vec{q}$, $\vec{q}[q'/q]$ to substitute the node q with the node q' in $\vec{q}$, we have that $\forall i : \vec{q}(i) \in \mathsf{Q}_i$ and finally we shall assume that the sets of nodes of the processes are mutually disjoint ($\forall i \neq j : \mathsf{Q}_i \cap \mathsf{Q}_j = \emptyset$).

The transitions of a timed system take the form

$$\langle \vec{q}_s, \sigma, \delta, \kappa \rangle \stackrel{\mathfrak{b}}{\Longrightarrow} \langle \vec{q}_t, \sigma', \delta', \kappa' \rangle$$

and the initial configurations are of the form $\langle \vec{q}_\circ, \sigma, \lambda r.0, \kappa_0 \rangle$ where $\vec{q}_\circ$ is the vector whose elements are the initial locations of the timed automata of the system and κ_0 maps each variable to the process that it belongs to ($\kappa_0(x) = \{p\}$ iff $x \in \mathbf{Var}_p$). The transition relation is given in Table 1.

The rule for the assignment, ensures that the guard is satisfied in the starting configuration and updates the mappings σ, δ, κ and the location of the process p_j and finally ensures that the invariant is satisfied in the resulting configuration. The behaviour $p_j : (\vec{x}, \vec{e})$ records

Table 1 Semantics for Timed Systems.

$$\langle \vec{q}_s, \sigma, \delta, \kappa\rangle \overset{p_j:(\vec{x},\vec{e})}{\Longrightarrow} \langle \vec{q}_t, \sigma', \delta', \kappa'\rangle \quad \text{if} \begin{cases} (q, g \to \vec{x} := \vec{e}\colon \vec{r}, q') \text{ is in } \mathsf{E}_j \\ [\![g]\!](\sigma,\delta) = \mathsf{tt} \\ \sigma' = \sigma[\vec{x} \mapsto [\![\vec{e}]\!]\sigma] \\ \delta' = \delta[\vec{r} \mapsto \vec{0}] \\ \kappa' = \kappa[\vec{x} \mapsto [\![\vec{e}]\!]\kappa] \\ \vec{q}_t = \vec{q}_s[q'/q] \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\vec{q}_t(i))]\!](\sigma',\delta') = \mathsf{tt} \end{cases}$$

$$\langle \vec{q}_s, \sigma, \delta, \kappa\rangle \overset{p_h:ch(\vec{x},\vec{e}):p_l}{\Longrightarrow} \langle \vec{q}_t, \sigma', \delta', \kappa'\rangle \quad \text{if} \begin{cases} h \neq l \\ (q_1, g_1 \to ch!\vec{e}\colon \vec{r}_1, q_1') \text{ is in } \mathsf{E}_h \\ (q_2, g_2 \to ch?\vec{x}\colon \vec{r}_2, q_2') \text{ is in } \mathsf{E}_l \\ \sigma' = \sigma[\vec{x} \mapsto [\![\vec{e}]\!]\sigma] \\ \delta' = (\delta[\vec{r}_1 \mapsto \vec{0}])[\vec{r}_2 \mapsto \vec{0}] \\ \kappa' = \kappa[\vec{x} \mapsto [\![\vec{e}]\!]\kappa] \\ \vec{q}_t = \vec{q}_s[q_1'/q_1][q_2'/q_2] \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\vec{q}_t(i))]\!](\sigma',\delta') = \mathsf{tt} \end{cases}$$

$$\langle \vec{q}, \sigma, \delta, \kappa\rangle \overset{\epsilon}{\Longrightarrow} \langle \vec{q}, \sigma, \delta', \kappa\rangle \quad \text{if} \begin{cases} \exists\, d > 0 : \delta' = \lambda r.\, \delta(r) + d, \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\vec{q}(i))]\!](\sigma,\delta') = \mathsf{tt} \end{cases}$$

that the process p_j is performing an assignment to the vector $\vec{x}$ using the vector $\vec{e}$, and κ' records the information flow that occurs due to this behaviour, by updating the writers of each variable $\vec{x}(i)$ with the writers of the expression $\vec{e}(i)$, where for a single expression e', $[\![e']\!]\kappa = \bigcup_{y \in fv(e')} \kappa(y)$ and $fv(e')$ is the set of free variables occuring in e'.

To understand the rule for the communication one could see it as an assignment of the form $\vec{x} := \vec{e}$ where $\vec{e}$ are the expressions which are used at the channel output action and $\vec{x}$ the variables that are used in the channel input action.

Finally, the delay rule only modifies the clock assignment with a delay d ensuring that the invariant is satisfied in the resulting configuration. The mapping κ remains the same since the delay action produces the empty behaviour ϵ.

▶ **Example 2.** To see how the semantics for the κ mapping works, return to Example 1 and consider the transition

$$\langle \vec{q}, \sigma, \delta, \kappa\rangle \overset{p_1:in_1(x,x_1):m}{\Longrightarrow} \langle \vec{q}[6/5], \sigma[x \mapsto \sigma(x_1)], \delta[v \mapsto 0], \kappa[x \mapsto \{p_1\}]\rangle$$

which corresponds to the communication between the the producer p_1 and the multiplexer m. We have that $\vec{q} = (1, 2, 5, 8, 3, 4)$, and let

$$\kappa = [x_1 \mapsto \{p_1\}, x_2 \mapsto \{p_2\}, x \mapsto \{m\}, y \mapsto \{m\}, z \mapsto \{d\}, z_1 \mapsto \{c_1\}, z_2 \mapsto \{c_2\}]$$

and the resulting mapping $\kappa[x \mapsto \{p_1\}]$ records that p_1 has written its value into the variable x, since there is an explicit flow from the variable x_1 to the variable x and x_1 has previously been written by p_1.

4 Time Dependent Policies in BTCTL

In this section, we present our behaviour based logic **BTCTL** which serves to specify time-dependent security policies for access control, based on the behaviours of the system. The access control policies can then be enforced statically before the execution of the system.

4.1 The Logic

The syntax of the **BTCTL** formulas ϕ is given by

$$\phi ::= g \mid set_1 \,\mathsf{rel}\, set_2 \mid \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2) \mid \phi_1 \wedge \phi_2 \mid \neg\phi$$

where

$$set ::= \underline{e} \mid W\,.$$

We have basic formulas which can be either a guard g, or relations between two sets of writers, $set_1 \,\mathsf{rel}\, set_2$, where $\mathsf{rel} = \{\subseteq, \supseteq\}$. The underlined set expression $\underline{e}$ denotes the set of writers of the expression e and $W \in \mathbb{P}(\mathbf{P})$ is a set of writers. We use the box operator $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ to speak about pre- and post-conditions whenever the non-empty behaviour $\mathfrak{b} \neq \epsilon$ happens. Informally speaking, a configuration γ will satisfy the $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ formula whenever for all of the system runs starting at γ, if a transition labelled with the behaviour $\mathfrak{b}$ occurs, then ϕ_1 should hold at the configuration before the transition and ϕ_2 at the configuration after it. As we will see shortly, the box operator will be the key formula to express access control policies. The $\neg\phi$ and $\phi_1 \wedge \phi_2$ cases are the usual ones. Finally, we sometimes write $\phi_1 \Rightarrow \phi_2$ for $\neg(\phi_1 \wedge \neg\phi_2)$.

▶ **Example 3.** Going back to Example 1, each of the variables has a time-dependent policy which specifies the maximum set of permitted writers of the variable. We are interested in the policies of the variables of the multiplexer, the demultiplexer and the two consumers:

$$\begin{aligned}
P_x &= (0 \leq t \wedge t < 5 \Rightarrow \underline{x} \subseteq \{p_1\}) \wedge \\
&\quad\; (5 \leq t \wedge t \leq 7 \Rightarrow \underline{x} \subseteq \{p_1, p_2\}) \wedge \\
&\quad\; (7 < t \wedge t \leq 10 \Rightarrow \underline{x} \subseteq \{p_2\})\,, \\
P_y &= \underline{y} \subseteq \{m\}\ , \\
P_z &= (0 \leq t \wedge t \leq 7 \wedge y = 1 \Rightarrow \underline{z} \subseteq \{p_1\}) \wedge \\
&\quad\; (5 \leq t \wedge t \leq 10 \wedge y = 2 \Rightarrow \underline{z} \subseteq \{p_2\})\,, \\
P_{z_1} &= \underline{z_1} \subseteq \{p_1\}\ , \\
P_{z_2} &= \underline{z_2} \subseteq \{p_2\}\ .
\end{aligned}$$

The first line of the policy for the variable x, expresses that whenever $t \in [0, 5)$, only the process p_1 is allowed to write data to x, while both p_1 and p_2 may write to x if $t \in [5, 7]$ and similarly to the first line, if $t \in (7, 10]$ then only p_2 can write to x. On the other hand, looking at the policy for the variable y, a write action to y is allowed only by the multiplexer. The rest of the policies can be explained accordingly.

We then perform the enforcement of the access control policies by checking the following formulas:

$$\begin{aligned}
\Phi_x &= \forall\Box_{p_1:in_1(x,x_1):m}(\mathsf{tt}, P_x) \wedge \forall\Box_{p_2:in_2(x,x_2):m}(\mathsf{tt}, P_x)\,, \\
\Phi_{y,z} &= \forall\Box_{m:ch((y,z),(1,x)):d}(\mathsf{tt}, P_y \wedge P_z) \wedge \forall\Box_{m:ch((y,z),(2,x)):d}(\mathsf{tt}, P_y \wedge P_z)\,, \\
\Phi_{z_1} &= \forall\Box_{d:out_1(z_1,z):c_1}(\mathsf{tt}, P_{z_1})\,, \\
\Phi_{z_2} &= \forall\Box_{d:out_2(z_2,z):c_2}(\mathsf{tt}, P_{z_2})\,.
\end{aligned}$$

Each of the formulas express that whenever someone is writing to the variable (or variables) appearing as a subscript, then the policy of the variable (or variables) is imposed as a post

condition. The variable x is accessed (someone is writing data to x) whenever p_1 and p_2 communicates with the multiplexer and thus we have to impose the policy of the variable x for both of those actions, while the variables y and z are being accessed whenever the multiplexer communicates with the demultiplexer and that happens with two communication actions. Similarly, we define the formula for the variables z_1 and z_2.

4.2 Semantics of BTCTL

The formal rules that define whenever a configuration γ satisfies a **BTCTL** formula ϕ are given below:

$$
\begin{array}{lll}
\gamma \models g & \text{iff} & \gamma = \langle \vec{q}, \sigma, \delta, \kappa \rangle \Rightarrow [\![g]\!](\sigma, \delta) \\
\gamma \models set_1 \,\mathsf{rel}\, set_2 & \text{iff} & \gamma = \langle \vec{q}, \sigma, \delta, \kappa \rangle \Rightarrow [\![set_1]\!]\kappa \,\mathsf{rel}\, [\![set_2]\!]\kappa \\
\gamma \models \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2) & \text{iff} & \forall \gamma_0 \overset{\mathfrak{b}_1}{\Rightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Rightarrow} .. \in \mathbf{Trace}_\gamma : \\
 & & \forall i \geq 1 : \mathfrak{b}_i = \mathfrak{b} \Rightarrow \gamma_{i-1} \models \phi_1 \text{ and } \gamma_i \models \phi_2 \\
\gamma \models \phi_1 \wedge \phi_2 & \text{iff} & \gamma \models \phi_1 \text{ and } \gamma \models \phi_2 \\
\gamma \models \neg\phi & \text{iff} & \gamma \not\models \phi
\end{array}
$$

A guard g is then satisfied by a configuration γ whenever g holds in γ. For the case of the set relation rel, γ satisfies it whenever $set_1 \,\mathsf{rel}\, set_2$ evaluates to true and we do that check by lifting the definition of $[\![.]\!]\kappa$ to set expressions, by $[\![\underline{e}]\!]\kappa = [\![e]\!]\kappa$ and $[\![W]\!]\kappa = W$. A configuration γ satisfies the box formula $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ whenever for all the execution paths that start from γ, if a behaviour $\mathfrak{b}'$ occurs and $\mathfrak{b}'$ is syntactically equal to $\mathfrak{b}$, then the pre-condition ϕ_1 has to hold at the configuration before the behaviour and the post-condition ϕ_2 at the configuration after it. The rest of the cases are the usual ones.

▶ **Example 4.** Consider now the prefix of an execution trace of the timed system from Example 1

$$pr = \gamma_0 \overset{p_1:in_1(x,x_1):m}{\Longrightarrow} \gamma_1 \overset{m:ch((y,z),(1,x)):d}{\Longrightarrow} \gamma_2 \overset{d:out_1(z_1,z):c_1}{\Longrightarrow} \gamma_3$$

where for the initial configuration $\gamma_0 = \langle \vec{q}_0, \sigma_0, \delta_0, \kappa_0 \rangle$, we have that $\vec{q}_0 = (1, 2, 5, 8, 3, 4)$, σ_0 is arbitrary, $\delta_0 = \lambda c.0$ and

$$\kappa_0 = [x_1 \mapsto \{p_1\}, x_2 \mapsto \{p_2\}, x \mapsto \{\, m \,\}, y \mapsto \{d\}, z \mapsto \{d\}, z_1 \mapsto \{\, c_1 \,\}, z_2 \mapsto \{c_2\}]$$

and for the rest of the configurations

$$
\begin{aligned}
&\gamma_1 = \langle \vec{q}_1, \sigma_1, \delta_1, \kappa_1 \rangle, \vec{q}_1 = \vec{q}_0[6/5], \sigma_1 = \sigma_0[x \mapsto \sigma_0(x_1)], \delta_1 = \delta_0[v \mapsto 0], \kappa_1 = \kappa_0[x \mapsto \{p_1\}] \\
&\gamma_2 = \langle \vec{q}_2, \sigma_2, \delta_2, \kappa_2 \rangle, \vec{q}_2 = \vec{q}_1[5/6][9/8], \sigma_2 = \sigma_1[y \mapsto 1, z \mapsto \sigma_1(x)], \delta_2 = \delta_1[r \mapsto 0], \\
&\qquad \kappa_2 = \kappa_1[y \mapsto \emptyset, z \mapsto \{p_1\}] \\
&\gamma_3 = \langle \vec{q}_3, \sigma_3, \delta_3, \kappa_3 \rangle, \vec{q}_3 = \vec{q}_1[8/9], \sigma_3 = \sigma_2[z_1 \mapsto \sigma_2(z)], \delta_3 = \delta_2, \kappa_3 = \kappa_2[z_1 \mapsto \{p_1\}].
\end{aligned}
$$

Now consider the formulas Φ_x, $\Phi_{y,z}$, Φ_{z_1} from Example 3 and to illustrate how the semantics work for the box operator, we will do the appropriate checks for those formulas on pr.

The formula Φ_x is the conjuction of two box operators, where for the first one because of the behaviour $p_1 : in_1(x, x_1) : m$ of the transition $\gamma_0 \overset{p_1:in_1(x,x_1):m}{\Longrightarrow} \gamma_1$, we have to check that $\gamma_1 \models P_x$ where

$$
\begin{aligned}
P_x = &(0 \leq t \wedge t < 5 \Rightarrow \underline{x} \subseteq \{p_1\}) \wedge \\
&(5 \leq t \wedge t \leq 7 \Rightarrow \underline{x} \subseteq \{p_1, p_2\}) \wedge \\
&(7 < t \wedge t \leq 10 \Rightarrow \underline{x} \subseteq \{p_2\}) .
\end{aligned}
$$

This check evaluates to true, since γ satisfies only the guard of the first line of the policy and $\kappa_1(x) = \{p_1\}$. For the transition $\gamma_1 \overset{m:ch((y,z),(1,x)):d}{\Longrightarrow} \gamma_2$, because of the formula $\Phi_{y,z}$ we have to check that $\gamma_2 \models P_y \wedge P_z$ where

$$P_y = \underline{y} \subseteq \{m\}\ ,$$
$$P_z = (0 \leq t \wedge t \leq 7 \wedge y = 1 \Rightarrow \underline{z} \subseteq \{p_1\}) \wedge$$
$$(5 \leq t \wedge t \leq 10 \wedge y = 2 \Rightarrow \underline{z} \subseteq \{p_2\})\,.$$

This check evaluates to true, since $\kappa_2(y) = \emptyset$ and γ_2 satisfies only the condition at the first line of the policy P_z and also $\kappa_2(z) = \{p_1\}$. Finally for the last transition $\gamma_2 \overset{d:out_1(z_1,z):c_1}{\Longrightarrow} \gamma_3$, because of the Φ_{z_1} formula, we have to check that $\gamma_3 \models P_z$, and this check evaluates to true, since $\kappa_3(z_1) = \{p_1\}$ and $P_z = \underline{z_1} \subseteq \{p_1\}$.

5 Reduction of BTCTL to TCTL$^+$

In this section, we perform a transformation of the original time system, and of the **BTCTL** formulas. The transformation is based on the work done in [16], where the action-based logic ATCTL (action-TCTL) is being reduced to TCTL [2]. A transformed formula produces a formula in **TCTL**$^+$, a logic based on TCTL and in the next section we show how a fragment of **TCTL**$^+$ can be handled by the model checker UPPAAL [30].

5.1 Behaviour Automata

A timed system $TS = (TA_i)_{i \leq n}$ yields a behaviour automaton $BA = (v_\circ, \mathsf{E}, \mathsf{I}, \mathsf{Q}, \mathsf{L})$, which is a kind of timed automaton in that it is the product automaton of the system, extended to contain *auxiliary vertices* that represent the actions of the system and a labelling function L that assigns to each vertex a property. A property is either a behaviour or a location vector of the system TS; auxiliary vertices of the system will be labeled with the behaviour that corresponds to the particular action of the vertex, while *genuine vertices* that represent locations of the system TS are labeled with a location vector. The initial vertex $v_\circ$ will be labeled with the initial location vector of the system $\vec{q}_\circ$. The behaviour automaton BA has the same set of variables as the timed system TS, while for the clock variables it has an extra clock t. Similarly to the timed automata, E is a finite set of edges, the mapping I imposes an invariant on each vertex and Q is the finite set of vertices.

The algorithm for constructing the edges E, the labelling functions I and L and the set of vertices $\mathsf{Q} = Q_{gen} \cup Q_{aux}$ ($Q_{gen} \cap Q_{aux} = \emptyset$) where Q_{gen} and Q_{aux} contain the genuine and auxiliary vertices respectively, is given in Figure 3.

In the first step, we create the genuine vertices and we label them with the invariant of the location vector that they represent; each of those vertices is inserted in Q_{gen}, which will be used in the next steps to create the auxiliary vertices.

In step 2 we create the auxiliary vertices and the edges that correspond to the assignment actions of the system. For each process p_i, we start looking at all of its assigment edges $(q_i, g \to \vec{x} := \vec{e}\colon \vec{r}, q_i') \in E_i$. For each one of those edges and for all the vertices $v_s \in Q_{gen}$ and $v_t \in Q_{gen}$, where the label of v_s, $\mathsf{L}(v_s)$ corresponds to a vector location where this assignment could have been performed and would have moved the system to the location $\mathsf{L}(v_t)$, we create the edges $(v_s, g \to \texttt{skip}\colon t, v)$ and $(v, \vec{x} := \vec{e}\colon \vec{r}, v_t)$, where v is a fresh auxiliary vertex; whereas in the construction of the product automaton one would have constructed only the edge $(v_s, g \to \vec{x} := \vec{e}\colon r, v_t)$. The auxiliary vertex v is labelled with the assignment behaviour

(1) let $Q_{gen} = \emptyset$; let $Q_{aux} = \emptyset$;
for all $\vec{q}$: create fresh v; let $\mathsf{L}(v) = \vec{q}$; let $\mathsf{I}(v) = \bigwedge_{i=1}^{n} \mathsf{I}_i(\vec{q}(i))$; insert v in Q_{gen}

(2) for all $(q_i, g \to \vec{x} := \vec{e}\colon \vec{r}, q_i') \in E_i$:

$$\text{for all } v_s \in Q_{gen},\ v_t \in Q_{gen} \text{ such that } \begin{cases} \mathsf{L}(v_s)(i) = q_i \\ \mathsf{L}(v_t)(i) = q_i' \\ \forall j : j \neq i : \mathsf{L}(v_s)(j) = \mathsf{L}(v_t)(j) \end{cases} :$$

create fresh v;
insert $(v_s, g \to \texttt{skip}\colon t, v)$ in E; insert $(v, \vec{x} := \vec{e}\colon \vec{r}, v_t)$ in E ;
let $\mathsf{L}(v) = p_i : (\vec{x}, \vec{e})$; let $\mathsf{I}(v) = (t = 0) \wedge \mathsf{I}(v_t)[\vec{e}/\vec{x}][\vec{0}/\vec{r}]$; insert v in Q_{aux}

(3) for all $(q_i, g_1 \to ch!\vec{e}\colon \vec{r}_1, q_i') \in E_i$ and $(q_j, g_2 \to ch?\vec{x}\colon \vec{r}_2, q_j') \in E_j$ such that $i \neq j$:

$$\text{for all } v_s \in Q_{gen},\ v_t \in Q_{gen} \text{ such that } \begin{cases} \mathsf{L}(v_s)(i) = q_i \wedge \mathsf{L}(v_s)(j) = q_j \\ \mathsf{L}(v_t)(i) = q_i' \wedge \mathsf{L}(v_t)(j) = q_j' \\ \forall l : l \neq i \wedge l \neq j : \mathsf{L}(v_s)(l) = \mathsf{L}(v_t)(l) \end{cases} :$$

create fresh v; let $g = g_1 \wedge g_2$; let $\vec{r} = \vec{r}_1\vec{r}_2$;
insert $(v_s, g \to \texttt{skip}\colon t, v)$ in E; insert $(v, \vec{x} := \vec{e}\colon \vec{r}, v_t)$ in E;
let $\mathsf{L}(v) = p_i : ch(\vec{x}, \vec{e}) : p_j$; let $\mathsf{I}(v) = (t = 0) \wedge \mathsf{I}(v_t)[\vec{e}/\vec{x}][\vec{0}/\vec{r}]$; insert v in Q_{aux}

(4) let $\mathsf{Q} = Q_{gen} \cup Q_{aux}$

Figure 3 The algorithm for constructing E, I, Q and L.

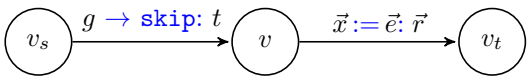


Figure 4 Edge construction of BA.

$p_i : (\vec{x}, \vec{e})$ and its invariant is being set to $(t = 0) \wedge \mathsf{I}(v_t)[\vec{e}/\vec{x}][\vec{0}/\vec{r}]$, to first ensure that the action of the edge leaving v will be performed instantenous and secondly that we can not get stuck at an auxiliary vertex. Figure 4 illustrates the construction and note that each auxiliary vertex v has *exactly one predecessor* and *exactly one successor*.

Similarly to step 2, in step 3 we construct the auxiliary vertices for the communication actions of the system and finally in step 4 we define the set Q.

5.2 Trace Equivalence

From the construction of the behaviour automaton BA, it is essential that every execution trace in the original system TS can be interpreted as an execution trace in the behaviour automaton BA and vice versa. Particularly, each transition in the system TS is equivalent to a single step transition (in the case of a delay) or a two-step transition (in the case of an action) in its behaviour automaton BA. To overcome the vagueness of this explanation we will later define an equivalence relation between execution traces of the system TS and the behaviour automaton BA.

First, we give the operational semantics of the behaviour automata in Table 2. The semantics is similar to the semantics of the timed automata, however now, the transitions are not labelled with behaviours.

Table 2 Semantics for Behaviour Automata.

$$\langle v_s, \sigma, \delta, \kappa\rangle \longrightarrow \langle v_t, \sigma', \delta', \kappa'\rangle \quad \text{if} \begin{cases} (v_s, g \to \vec{x} := \vec{e}\colon \vec{r}, v_t) \text{ is in } \mathsf{E} \\ [\![g]\!](\sigma, \delta) = \mathsf{tt} \\ \sigma' = \sigma[\vec{x} \mapsto [\![\vec{e}]\!]\sigma] \\ \delta' = \delta[\vec{r} \mapsto \vec{0}] \\ \kappa' = \kappa[\vec{x} \mapsto [\![\vec{e}]\!]\kappa] \\ [\![\mathsf{I}(v_t)]\!](\sigma', \delta') = \mathsf{tt} \end{cases}$$

$$\langle v, \sigma, \delta, \kappa\rangle \longrightarrow \langle v, \sigma, \delta', \kappa\rangle \quad \text{if} \begin{cases} \exists\, d > 0 : \delta' = \lambda r.\, \delta(r) + d, \\ [\![\mathsf{I}(v)]\!](\sigma, \delta') = \mathsf{tt} \end{cases}$$

Now let γ and γ' to be two configurations of a timed system TS and its behaviour automaton BA respectively. We define the relation $\cong$: $\mathbf{Config}_{TS} \times \mathbf{Config}_{BA} \to \{\mathsf{tt}, \mathsf{ff}\}$ to be

$$\langle \vec{q}, \sigma, \delta, \kappa\rangle \cong \langle v, \sigma', \delta', \kappa'\rangle \quad \text{iff} \quad \begin{array}{l} \vec{q} = \mathsf{L}(v)\,, \\ \sigma = \sigma'\,, \\ \forall r \in \mathbf{Clock} : \delta(r) = \delta'(r)\,, \\ \kappa = \kappa'\,, \end{array}$$

where we recall that **Clock** is the set of the clocks appearing in the system TS and thus the clock t of the behaviour automaton BA is *not* included in **Clock**. It is straightforward by the definition of $\cong$ that configurations of the system TS can only be related with configurations that correspond to genuine vertices in the behaviour automaton BA.

For the behaviour automata BA, we define a *macro transition* t to be a single step delay transition $\gamma'_s \longrightarrow \gamma'_t$ or a two-step transition $\gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t$, where γ_{aux} is an *auxiliary configuration* (a configuration that corresponds to an auxiliary vertex) and γ_s and γ_t are *genuine configurations* (configurations that correspond to genuine vertices). We then lift the definition of $\cong$ to single step transitions of the system TS and macro transitions of the behaviour automaton BA as

$$\gamma_s \overset{\epsilon}{\Longrightarrow} \gamma_t \cong \gamma'_s \longrightarrow \gamma'_t \text{ iff } \begin{cases} \gamma_s \cong \gamma'_s \\ \gamma_t \cong \gamma'_t \end{cases}$$

$$\gamma_s \overset{\mathfrak{b}}{\Longrightarrow} \gamma_t \cong \gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t \text{ iff } \begin{cases} \gamma_s \cong \gamma'_s \\ \gamma_t \cong \gamma'_t \\ \gamma_{aux} = \langle v, \sigma, \delta, \kappa\rangle \Rightarrow \mathfrak{b} = \mathsf{L}(v) \end{cases}$$

Now for each genuine configuration γ' of the BA, we have that every execution trace $tr' = \gamma'_0 \longrightarrow \gamma'_1 .... \in \mathbf{Trace}_{\gamma'}$ of γ', with length greater than 0, can be parsed as a *macro transition trace* $T_{tr'} = t'_1 t'_2 t'_3 .....$ where each t'_j is a macro transition. For example the finite execution trace $\gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma_{aux_1} \longrightarrow \gamma'_2 \longrightarrow \gamma_{aux_2} \longrightarrow \gamma'_3 \longrightarrow \gamma'_4$, which is a sequence of a delay, action(two-step), action(two-step), delay, will produce the macro transition trace $t'_1 t'_2 t'_3 t'_4$ where $t'_1 = \gamma'_0 \longrightarrow \gamma'_1$, $t'_2 = \gamma'_1 \longrightarrow \gamma_{aux_1} \longrightarrow \gamma'_2$, $t'_3 = \gamma'_2 \longrightarrow \gamma_{aux_2} \longrightarrow \gamma'_3$ and $t'_4 = \gamma'_3 \longrightarrow \gamma'_4$.

Similarly to the macro transition traces, for each configuration γ of the system TS, we can write each nonzero-length execution trace, $tr = \gamma_0 \overset{\mathfrak{b}_1}{\Longrightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Longrightarrow} \gamma_2 ... \in \mathbf{Trace}_\gamma$ of γ, as a *transition trace* $T_{tr} = t_1 t_2 ...$ where $t_i = \gamma_{i-1} \overset{\mathfrak{b}_i}{\Longrightarrow} \gamma_i$ (for all $i \geq 1$).

Finally we lift the definition of $\cong$ to execution traces of length $n > 0$, that start in genuine configurations inside the timed system TS and its behaviour automaton BA as

$$tr \cong tr' \text{ iff } \begin{cases} T_{tr} \text{ and } T_{tr'} \text{ have the same length} \\ \forall i \geq 1 : T_{tr}(i) \cong T_{tr'}(i) \end{cases}$$

$tr \cong tr'$, then results to true if and only if the transition trace T_{tr} of tr and the macro transition trace $T_{tr'}$ of tr' have the same length and they are equivalent stepwise.

The following fact follows from the method of constructing a behaviour automaton and states that equivalent configurations in the timed system TS and its behaviour automaton BA, produce equivalent execution traces.

▶ **Fact 5.** *For every timed system TA, its behaviour automaton BA and two configurations γ and γ' such that $\gamma \cong \gamma'$ we have that:*

- $\forall tr \in \boldsymbol{Trace}_{\gamma} : \exists tr' \in \boldsymbol{Trace}_{\gamma'} : tr \cong tr'$,
- $\forall tr' \in \boldsymbol{Trace}_{\gamma'} : \exists tr \in \boldsymbol{Trace}_{\gamma} : tr \cong tr'$.

5.3 TCTL$^+$

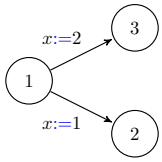
For the behaviour automata, we define a new logic called **TCTL**$^+$ patterned after TCTL [2], and the syntax of a **TCTL**$^+$ formula ψ is given by

$$\psi ::= prop \mid g \mid set_1 \text{ rel } set_2 \mid \forall\Box\psi \mid \exists(\psi_1 U \psi_2) \mid \neg\psi \mid \psi_1 \wedge \psi_2 \,.$$

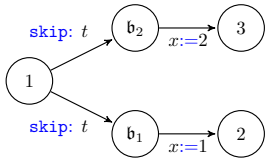
The basic formula *prop* is a proposion which is either a behaviour or a location vector and it holds in a configuration if its vertex is labelled with *prop*; the rest of the basic formulas are the same as in **BTCTL** . The $\forall\Box\psi$ formula holds in a configuration if for all of its execution traces, ψ holds in all the configurations of the trace, while for the $\exists(\psi_1 U \psi_2)$ to hold, it is sufficient that there exists an execution trace where ψ_1 holds for a prefix of the trace and eventually ψ_2 also holds. The rest of the operators are the same as in **BTCTL** . The formal semantics of the **TCTL**$^+$ is given by:

$\gamma' \models prop$	iff	$\gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow \mathsf{L}(v) = prop$
$\gamma' \models g$	iff	$\gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow [\![g]\!](\sigma, \delta)$
$\gamma' \models set_1 \text{ rel } set_2$	iff	$\gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow [\![set_1]\!]\kappa \text{ rel } [\![set_2]\!]\kappa$
$\gamma' \models \forall\Box\psi$	iff	$\forall \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma'_2 .... \in \mathbf{Trace}_{\gamma'} : \forall i \geq 0 : \gamma'_i \models \psi$
$\gamma' \models \exists(\psi_1 U \psi_2)$	iff	$\exists \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma'_2 .... \in \mathbf{Trace}_{\gamma'} :$
		$\exists i : \gamma'_i \models \psi_2$ and $\forall j < i : \gamma'_j \models \psi_1$
$\gamma' \models \psi_1 \wedge \psi_2$	iff	$\gamma' \models \psi_1$ and $\gamma' \models \psi_2$
$\gamma' \models \neg\psi$	iff	$\gamma' \not\models \psi$

Our goal is to transform a **BTCTL** formula ϕ into a **TCTL**$^+$ formula ψ and then show that for two equivalent configurations γ and γ' of a timed system TS and its behaviour automaton BA respectively, checking the formula ϕ in γ it is sufficient to check the transformed formula ψ in γ' and vice versa. We perform the transformation of the formulas using a function $\mathcal{T}[\![.]\!]$



(a) The timed automaton of the process p



(b) The behaviour automaton of p

as follows

$$\begin{aligned}
\mathcal{T}[\![g]\!] &= g\,,\\
\mathcal{T}[\![set_1 \,\mathsf{rel}\, set_2]\!] &= set_1 \,\mathsf{rel}\, set_2\,,\\
\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)]\!] &= \forall\Box(\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])))\,,\\
\mathcal{T}[\![\phi_1 \wedge \phi_2]\!] &= \mathcal{T}[\![\phi_1]\!] \wedge \mathcal{T}[\![\phi_2]\!]\,,\\
\mathcal{T}[\![\neg\phi]\!] &= \neg\mathcal{T}[\![\phi]\!]\,.
\end{aligned}$$

For the special cases $\forall\Box_{\mathfrak{b}}(\mathtt{tt},\phi_2)$ (ϕ_2 is not $\mathtt{tt}$) and $\forall\Box_{\mathfrak{b}}(\phi_1,\mathtt{tt})$ (ϕ_1 is not $\mathtt{tt}$) we shall omit the transformed formula that corresponds to the trivial formula $\mathtt{tt}$, by writting $\mathcal{T}[\![\forall\Box_b(\mathtt{tt},\phi_2)]\!] = \forall\Box(\mathfrak{b} \Rightarrow \mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))$ for the first case and $\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1,\mathtt{tt})]\!] = \forall\Box(\mathfrak{b} \Rightarrow \mathcal{T}[\![\phi_1]\!])$ for the second case. Finally, we shall assume that formulas in the pre-condition of the $\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)$ are not nested. To justify this assumption consider the following example

▶ **Example 6.** Consider the timed automaton of a process p (Figure 5a) with a variable x and a clock r, and its behaviour automaton BA (Figure 5b), where $\mathfrak{b}_1 = p : (x, 1)$ and $\mathfrak{b}_2 = p : (x, 2)$ are the behaviours of the actions $x{:=}1$ and $x{:=}2$ respectively, and all the location invariants in the timed automaton of p are $\mathtt{tt}$.

Now let $\phi = \forall\Box_{\mathfrak{b}_1}(\forall\Box_{\mathfrak{b}_2}(\mathtt{tt}, x = 1), \mathtt{tt})$ and observe that every initial configuation of the process p does not satisfy ϕ, whereas every initial configuration of the behaviour automaton does satisfy the transformed formula $\mathcal{T}[\![\phi]\!] = \forall\Box(\mathfrak{b}_1 \Rightarrow (\forall\Box(\mathfrak{b}_2 \Rightarrow \exists(\mathfrak{b}_2\, U(\neg\mathfrak{b}_2 \wedge x = 1)))))$.

Since the proposed formula transformation is sufficient to express and enforce access control policies of our interest we leave the development of transformations that support the entire **BTCTL** as future work.

Finally, we state the correctness of the function $\mathcal{T}[\![.]\!]$ with the following theorem

▶ **Theorem 7.** *For a timed system TS, its behaviour automaton BA, a **BTCTL** formula ϕ and for every configuration γ and γ' of TS and BA respectively, we have that if $\gamma \cong \gamma'$ then*

$$\gamma \models \phi \text{ iff } \gamma' \models \mathcal{T}[\![\phi]\!]$$

The proof of Theorem 7 can be found in Appendix A.

5.4 Reduction Complexity

We give a computation bound for the algorithm of Figure 3, that given a timed system $TS = (TA_i)_{i\leq n}$ constructs the behaviour automaton $BA = (v_\circ, \mathsf{E}, \mathsf{I}, \mathsf{Q}, \mathsf{L})$. Assuming that the computation time of all the simple operations (creation of fresh vertices, setting of invariants e.t.c) is constant, we have that : let $K = |\mathsf{Q}_1| + ... + |\mathsf{Q}_n|$ and $E = |\mathsf{E}_1| + ... + |\mathsf{E}_n|$ then the first part of the algorithm is bounded by K^n. The second part iterates over the assignement edges and all the pairs of the auxiliary vertices and that is bounded by $E \times K^{2n} \times n$, where

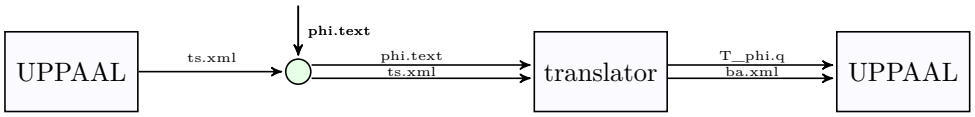


Figure 6 Architecture of the Translator.

n corresponds to the computation bound of checking the third condition of the branch of the for-loop. Similarly to the second part of the algorithm the third part is bounded by $E^2 \times K^{2n} \times n$ and therefore for the total sum of those bounds we obtain a complexity of $O(E^2 \times K^{2n} \times n)$. Finally, for a **BTCTL** formula ϕ the complexity of the transformation $\mathcal{T}[\![\phi]\!]$ is linear to the size of ϕ.

6 The Translator

We have implemented a translator in Java that works together with the model checker UPPAAL version 4.0 [30]. Figure 6 depicts the architecture of the translator.

UPPAAL is using a graphical interface in which one can model (draw) a system of timed automata. We first do that and next UPPAAL saves it as a file in the eXtensible Markup Language (XML) [33]; the xml file together with a text file that contains the desired property ϕ that we want to check, are being passed to the translator. The translator parses the two files and produces an xml file which contains the behaviour automaton of the system together with a UPPAAL query file that includes the property $\mathcal{T}[\![\phi]\!]$. The two files are imported to UPPAAL and then one can check if the desired property holds.

Since UPPALL does not allow nested formulas nor supports the operator $\exists\phi_1 U\phi_2$, we had to find a workaround for some of the transformed formulas. The guards g are translated directly; for the set_1 rel set_2, we model a set as a bit array since UPPALL supports multidimensional integer arrays and then we check the bit version of the relation rel . In case of the $\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)]\!] = \forall\Box(\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))))$, UPPAAL allows labelling a vertex with a string (the name of the vertex) and thus auxiliary vertices with label $\mathfrak{b}$ have as a name a string that corresponds to the behaviour $\mathfrak{b}$. For the part $\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])$ we annotate the outgoing edges of the auxiliary vertices with an assignment to a fresh variable a that works as a switch. We switch on by $a := 1$, only when we leave the auxiliary vertex, and we switch off by $a := 0$, whenever we leave the successor of the auxiliary vertex. Thus the formula $\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])$ is transformed into the formula $a = 1 \Rightarrow \mathcal{T}[\![\phi_2]\!]$.

Finally, since the mapping κ is not part of the timed automata of UPPAAL, we first enumerate each variable and each process of the system and we then model κ as a two-dimensional array, whose first index corresponds to a variable and whose second to a process. For instance, if a variable x is enumerated with 1 and $\kappa(x) = \{p\}$, where p is a process of the system and p is enumerated with 2, then $\kappa[1][2] = 1$, while for any other index $j \neq 2$, $\kappa[1][j] = 0$, modelling in that way that only p has written data in x. The edges of the automaton are also annotated with assignments to κ to capture the updates to it whenever the system performs an action.

7 Conclusions

We have successfully shown how to enforce access control policies on Systems of Timed Automata using a behaviour-based logic. The logic allows specification of time, data's content and information flow dependent security policies, an essential need in the modern world of

cyberphysical systems. We have developed a sound reduction of a substantial fragment of our logic to a logic based on TCTL [2], so that the model checking of the formulas can be performed by existing model checkers such as UPPAAL [30]. We implemented a translator which performs the reduction and together with UPPAAL it enforces access control policies. Finally, we illustrated our development using an example from the aerospace industry, where ensuring data's integrity is a life critical goal.

There are several ways in which we can extend our work. We are currently exploring how our development can be extended to capture more complex information flows such as *implicit* flows [31]. We have shown in [24] that the time aspect, as well as the non-deterministic semantics of Timed Automata, poses a challenge for that.

We are considering extensions to our logic that allow expressing richer access control policies and also how to develop a reduction which supports the entire syntax of the **BTCTL** logic. Another possibility is to explore new algorithms for determining if a formula of our logic holds in a timed system rather than reducing the formula to current TCTL-based logics.

References

1 Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007.

2 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

4 David A. Basin, Matús Harvan, Felix Klaedtke, and Eugen Zalinescu. Monitoring usage-control policies in distributed systems. *TIME*, pages 88–95, 2011.

5 Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

6 Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

7 Hsing-Chung Chen, Shiuh-Jeng Wang, Jyh-Horng Wen, Yung-Fa Huang, and Chung-Wei Chen. A generalized temporal and spatial role-based access control model. *JNW*, 5(8):912–920, 2010.

8 Carlo Combi, Roberto Posenato, Luca Viganò, and Matteo Zavatteri. Access controlled temporal networks. *ICAART (2)*, pages 118–131, 2017.

9 Carlo Combi, Luca Viganò, and Matteo Zavatteri. Security constraints in temporal role-based access-controlled workflows. *CODASPY*, pages 207–218, 2016.

10 Henry DeYoung, Deepak Garg, and Frank Pfenning. An authorization logic with explicit time. *CSF*, pages 143–165, 2008.

11 Sabrina De Capitani di Vimercati, Pierangela Samarati, and Ravi Sandhu. Access control. *Computing Handbook, 3rd ed.*, 1:47:1–25, 2014.

12 David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–264, 2001.

13 Emsaieb Geepalla, Behzad Bordbar, and Kozo Okano. Verification of spatio-temporal role based access control using timed automata. *NESEA*, pages 1–6, 2012.

14 Yong-Zhong He, Zhen Han, and Ye Du. Context active rbac and its applications. *ISECS*, pages 1041–1044, 2008.

15 Xuezhen Huang, Jiqiang Liu, and Zhen Han. A privacy-aware access model on anonymized data. *INTRUST*, pages 201–212, 2014.

16 David N. Jansen and Roel Wieringa. Extending ctl with actions and real time. *J. Log. Comput.*, 12(4):607–621, 2002.

17 Xin Jin, Ram Krishnan, and Ravi S. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. *DBSec*, pages 41–45, 2012.

18 James Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.

19 M. Fahim Ferdous Khan and Ken Sakamura. A discretionary delegation framework for access control systems. *OTM Conferences*, pages 865–882, 2016.

20 Samrat Mondal and Shamik Sural. Security analysis of temporal-rbac using timed automata. *IAS*, pages 37–40, 2008.

21 Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. Security analysis of gtrbac and its variants using model checking. *Computers and Security*, 30(2-3):128–147, 2011.

22 Kevin Mueller, Michael Paulitsch, Sergey Tverdyshev, and Holger Blasum. Mils-related information flow control in the avionic domain: A view on security-enhancing software architectures. *DSN Workshops*, pages 1–6, 2012.

23 Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *ACM Symposium on Operating System Principles, SOSP 1997*, pages 129–142. ACM, 1997.

24 Flemming Nielson, Hanne Riis Nielson, and Panagiotis Vasilikos. Information flow for timed automata. *Accepted for Springer Lecture Notes in Computer Science*, 2017.

25 Hanne Riis Nielson and Flemming Nielson. Content dependent information flow control. *J. Log. Algebr. Meth. Program.*, 87:6–32, 2017.

26 Martin Leth Pedersen, Michael Hedegaard Sørensen, Daniel Lux, Ulrik Nyman, and René Rydhof Hansen. The timed decentralised label model. *NordSec*, pages 27–43, 2015.

27 Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies and complex constraints. *NDSS*, 2001.

28 Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 1993.

29 Ravi S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Com. Mag.*, 1996.

30 UPPALL. `http://www.uppaal.com/index.php?sida=200&rubrik=95`.

31 Dennis M. Volpano, Geoffrey Smith, and Cynthia E. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

32 OASIS eXtensible Access Control Markup Language. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

33 eXtensible Markup Language(XML) . `https://www.w3.org/XML/`.

34 Wenrong Zeng, Yuhao Yang, and Bo Luo. Content-based access control: Use data content to assist access control for large-scale content-centric databases. *BigData Conference*, pages 701–710, 2014.

A Proof of Theorem 7

Proof. The proof proceeds by structural induction on ϕ. The base cases are trivial since $\mathcal{T}[\![\phi]\!] = \phi$, the formula ϕ does not include any constraint about the clock t, and $\gamma \cong \gamma'$.

- The case $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$.
  Assume that $\gamma \models \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ and thus by definition:

$$\forall \gamma_0 \overset{\mathfrak{b}_1}{\Rightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Rightarrow} .. \in \mathbf{Trace}_\gamma : \forall i \geq 1 : \mathfrak{b}_i = \mathfrak{b} \Rightarrow \gamma_{i-1} \models \phi_1 \text{ and } \gamma_i \models \phi_2 \,. \tag{1}$$

Now take arbritary trace $tr' = \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow ... \in \mathbf{Trace}_{\gamma'}$, where $\gamma' = \gamma'_0$ and prove that

$$\forall j \geq 0 : \gamma'_j \models \mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))) \,.$$

Now if tr' has length 0 then $tr' = \gamma'$ and the proof is trivial since γ' is a genuine configuration and thus $\gamma' \not\models \mathfrak{b}$. Similarly, if tr' has length greater than 0 and γ'_j is a genuine configuration then the proof holds. Now if γ'_j is an auxiliary configuration, consider the macro tranistion trace $T_{tr'} = t'_1 t'_2 ...$ of the trace tr' and let t'_i be the macro tranistion which corresponds to the transition in which γ'_j is being involved and thus we have that $T_{tr'}(i) = \gamma'_{j-1} \longrightarrow \gamma'_j \longrightarrow \gamma'_{j+1}$. Next, let $\gamma'_j = \langle v, \sigma, \delta, \kappa\rangle$ and using Fact 5 we have that $\exists tr \in \mathbf{Trace}_\gamma : tr \cong tr'$ and thus

$$\begin{aligned}
& \forall h \geq 1 : T_{tr}(h) \cong T_{tr'}(h) \\
\Rightarrow\quad & T_{tr}(i) \cong T_{tr'}(i) \\
\Leftrightarrow\quad & \gamma_{i-1} \overset{\mathfrak{b}_i}{\Longrightarrow} \gamma_i \cong \gamma'_{j-1} \longrightarrow \gamma'_j \longrightarrow \gamma'_{j+1} \\
\Leftrightarrow\quad & \gamma_{i-1} \cong \gamma'_{j-1} \text{ and } \gamma_i \cong \gamma'_{j+1} \text{ and } \mathsf{L}(v) = \mathfrak{b}_i
\end{aligned} \tag{2}$$

Now if $\gamma'_j \not\models \mathfrak{b}$ then the proof is trivial. Otherwise, because of (2) ($\mathsf{L}(v) = \mathfrak{b}_i$) we have that also $\mathfrak{b}_i = \mathfrak{b}$ and using (1) we have that $\gamma_{i-1} \models \phi_1$ and $\gamma_i \models \phi_2$. Next, using (2) ($\gamma_{i-1} \cong \gamma'_{j-1}$) and our induction hypothesis we have also that $\gamma'_{j-1} \models \mathcal{T}[\![\phi_1]\!]$ and since ϕ_1 does not contain any nested formulas we also have that $\gamma'_j \models \mathcal{T}[\![\phi_1]\!]$ as required. Finally, using (2) ($\gamma_i \cong \gamma'_{j+1}$) and our induction hypothesis we have also that $\gamma'_{j+1} \models \mathcal{T}[\![\phi_2]\!]$ and thus $\gamma'_j \models \exists(\mathfrak{b}\, U\, (\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))$ as required.
For the other direction now assume that $\gamma' \models \forall\Box\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])))$ and thus

$$\forall \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma'_2 .... \in \mathbf{Trace}_{\gamma'} : \forall i \geq 0 : \gamma'_i \models \mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))) \tag{3}$$

and take arbitrary trace $tr = \gamma_0 \overset{\mathfrak{b}_1}{\Longrightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Longrightarrow} ... \in \mathbf{Trace}_\gamma$, where $\gamma_0 = \gamma$ and prove that

$$\forall j \geq 1 : \mathfrak{b}_j = \mathfrak{b} \Rightarrow \gamma_{j-1} \models \phi_1 \text{ and } \gamma_j \models \phi_2 \,.$$

For the cases where the length of tr is 0 or $\mathfrak{b}_j \neq \mathfrak{b}$ then the proof is trivial. Therefore take j such that $\mathfrak{b}_j = \mathfrak{b}$ and consider the transition trace $T_{tr} = t_1 t_2 ....$ of the trace tr and thus, using Fact 5 we have that $\exists tr' \in \mathbf{Trace}_{\gamma'} : tr \cong tr'$ and consequently

$$\begin{aligned}
& \forall h \geq 1 : T_{tr}(h) \cong T_{tr'}(h) \\
\Rightarrow\quad & T_{tr}(j) \cong T_{tr'}(j) \\
\Leftrightarrow\quad & \gamma_{j-1} \overset{\mathfrak{b}_j}{\Longrightarrow} \gamma_j \cong \gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t \\
\Leftrightarrow\quad & \gamma_{j-1} \cong \gamma'_s \text{ and } \gamma_j \cong \gamma'_t \text{ and if } \gamma_{aux} = \langle v, \sigma, \delta, \kappa\rangle \text{ then } \mathsf{L}(v) = \mathfrak{b}_j \,.
\end{aligned} \tag{4}$$

Therefore because of (4) ($\mathsf{L}(v) = \mathfrak{b}_j$) and (3) we have that $\gamma_{aux} \models \mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\, U\, (\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))$ and thus since ϕ_1 does not contain any nested formulas, $\gamma'_s \models \mathcal{T}[\![\phi_1]\!]$ and $\gamma'_t \models \mathcal{T}[\![\phi_2]\!]$; but then using (4) ($\gamma_{j-1} \cong \gamma'_s$ and $\gamma_j \cong \gamma'_t$) and our induction hypothesis we get the required result.

The cases $\phi_1 \wedge \phi_2$ and $\neg\phi$ can be proved straightforwardly using structural induction on ϕ_1, ϕ_2 and ϕ. ◀